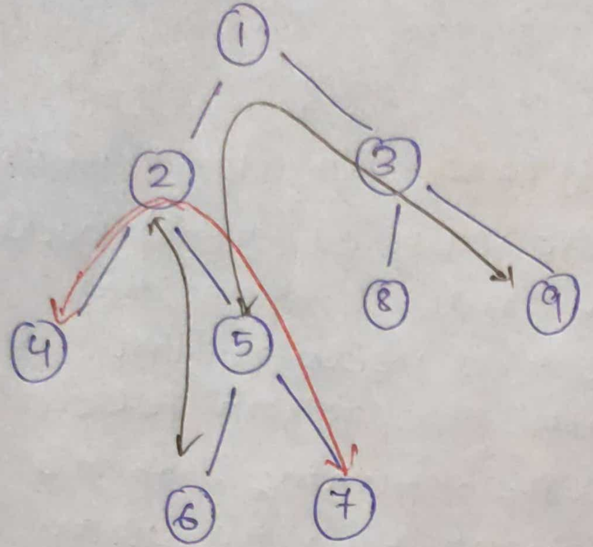
Lowest Common Ancestor :-



lca(4,7) = 2

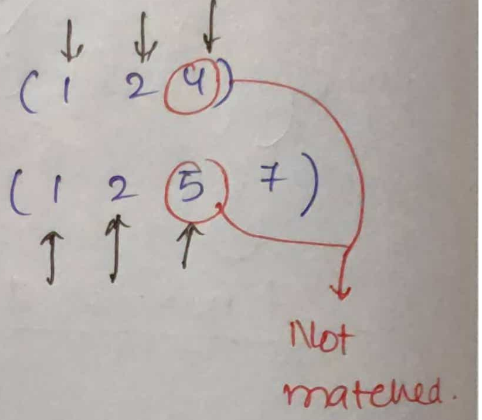
lca(5,9) = 1

lca(2,6) = 2.

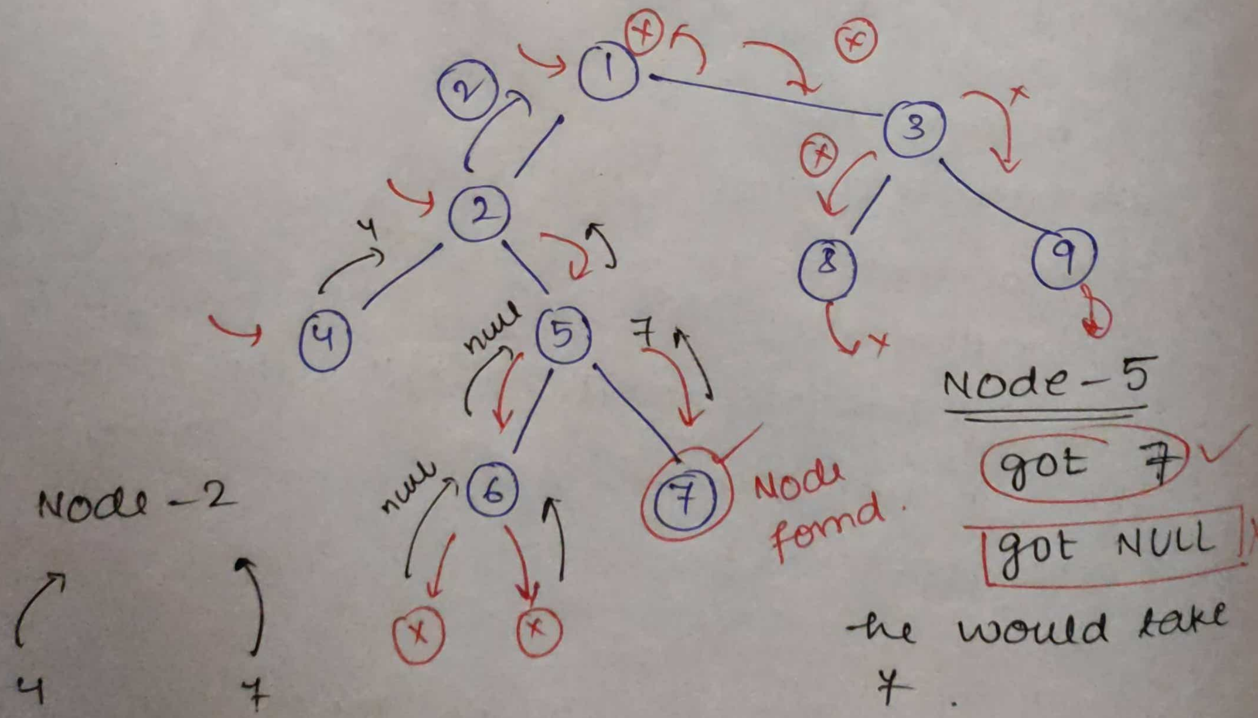
Brute Force :- node = 4

node = 7

check nodes val & the last value that matched.



We will follow the BFS traversal technique. (recursive)



Node-5

got 7 ✓

got NULL ✗

he would take 7.

↳ If both are not NULL that means we got our LCA.

Node-1 → got 2 & got NULL

taken

done!!

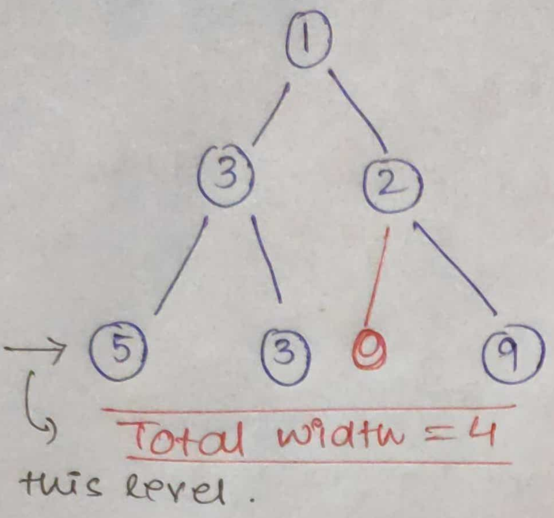
code:-

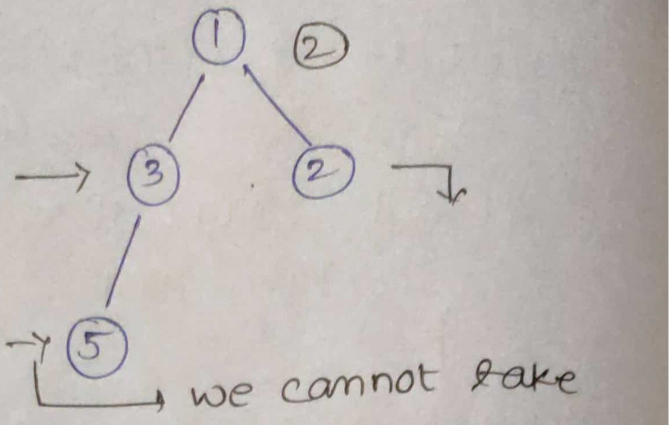
```
TreeNode * LCA(TreeNode *root, TreeNode *p,
                         TreeNode *q) {

   if(root==NULL || root==p || root==q) {
         return root;
    }
  TreeNode * left = LCA(root->left, p,q);
  TreeNode *right = LCA (root->right,p,q);

  // result
     if(left==NULL) {
           return right;
     }
    else if (right==NULL) {
           return left;
      }
     else { // both left & right != NULL
                       therefore, result found!
          return root;
     }
  }
```

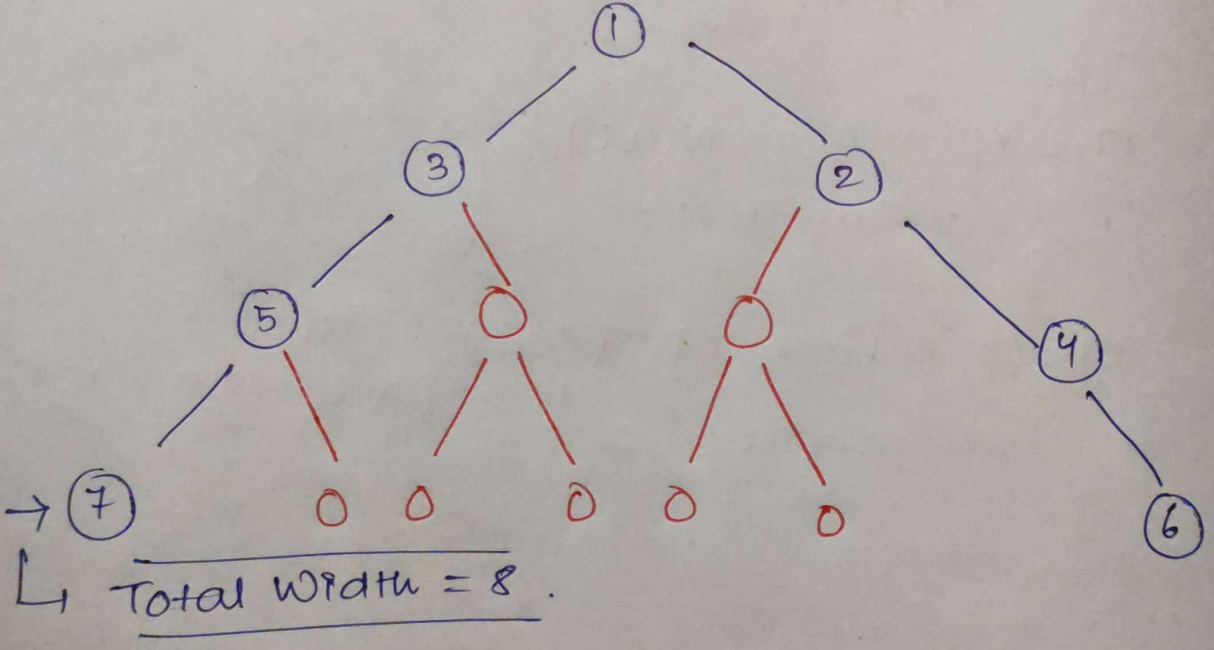
Maximum Width of a Binary Tree:-

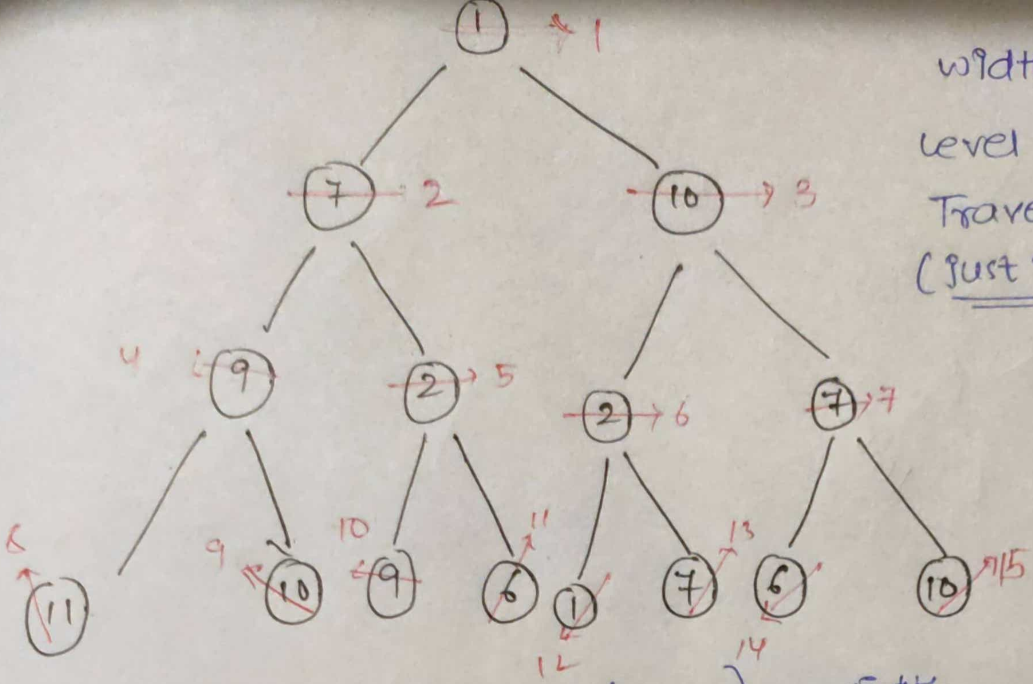




we cannot take this level as 2 doesn't have any node.

Width :- Number of nodes in a level between any 2 nodes.





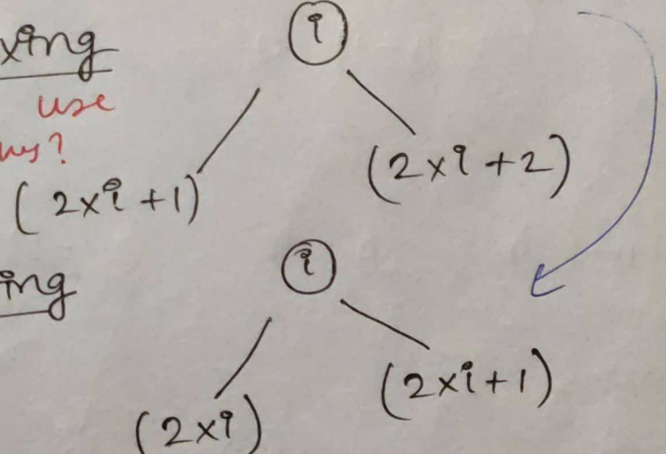
width → level

level order

Traversal

(just index them)

(last node − ~~first~~ node + 1) → width

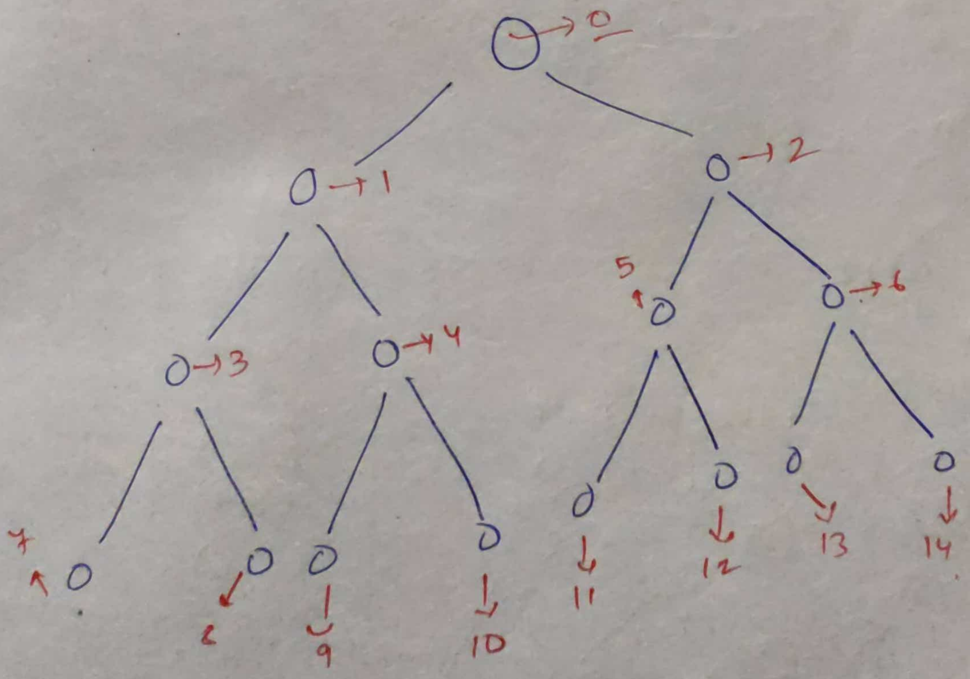


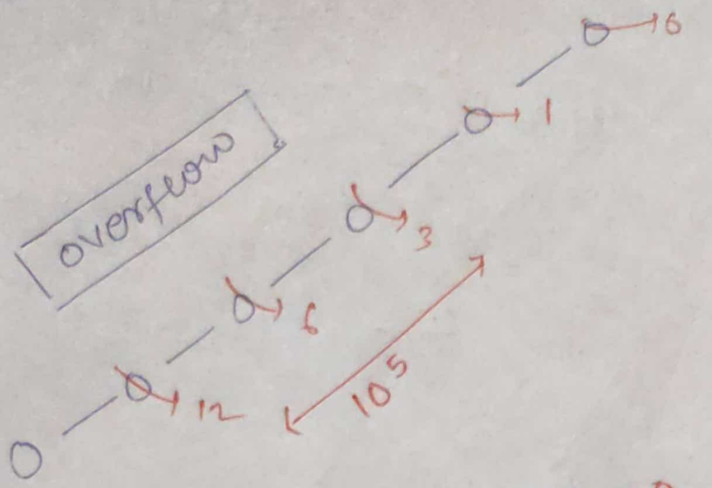
0-based indexing

but we cannot use this logic → why?

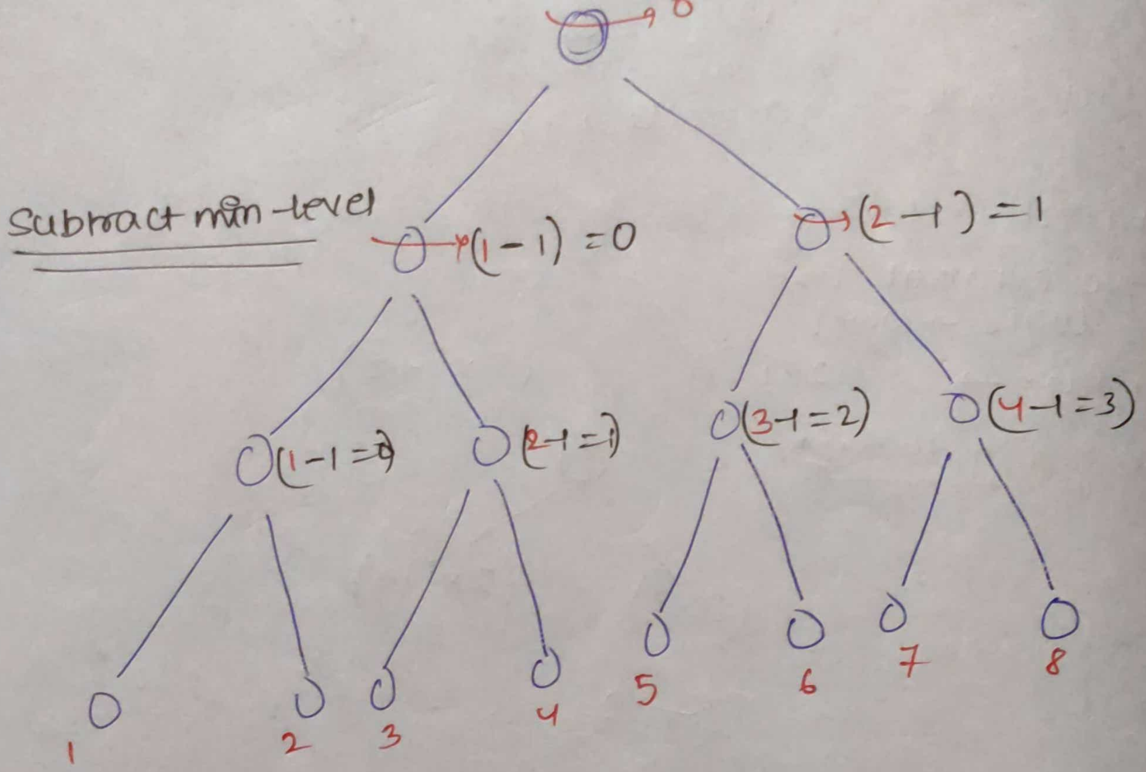
Next Page →

1-based indexing



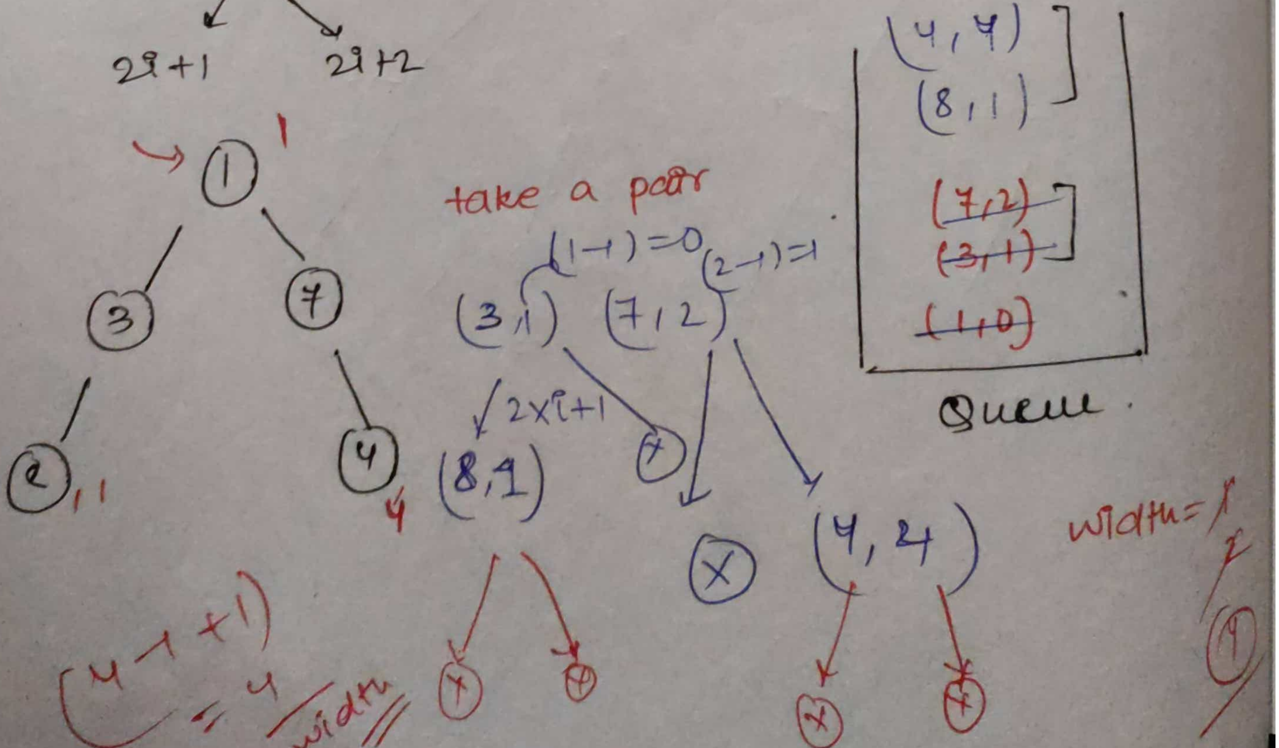


Subtract min-level



$i = \underline{\underline{(i - min)}}$

$2i+1 \quad 2i+2$



Code :-

```
int widthOfBT (TreeNode * root){
    if (!root) return 0;
    int ans=0;
   queue<pair< TreeNode *, int>> q;
   q.push ({root,0});
   while(!q.empty()) {
       int size = q.size();
       int mmin = q.front().second
       int first, last;
      for (int i=0; i<size ;i++){
          int cur_id= q.front().second - mmin;
          TreeNode *node = q.front().first;
          q.pop();
         if (i==0) first = cur_id;
        if(node->left)
              q.push({node->left, cur_id*2+1});
        if(node->right)
              q.push({node->right, cur_id*2+2});
      }
    ans= max(ans, last-first+1);
  }
  return ans;
}
```